13장. 스코프

13장. 스코프

13.1 스코프란?

1. **스코프**(scope) - 자바스크립트 포함한 모든 프로그래밍 언어의 기본적이며 중요한 개념

2. 자바스크립트의 스코프, 다른 언어의 스코프와 구별되는 특징 有

3. 함수의 매개변수 : 함수 몸체 내부에서만 참조할 수 있고, 외부에서는 참조 불가능

   ➡️ 매개변수 참조할 수 있는 유효범위, 즉 **매개변수의 스코프가 함수 몸체 내부로 한정**되기 때문

   [예제 13-01]

   ```
   function add(x, y){
       // 매개변수는 함수 몸체 내부에서만 참조할 수 있다.
       // 즉, 매개변수의 스코프(유효범위)는 함수 몸체 내부다.
       console.log(x, y);   // 2 5
       return x + y;
   }

   add(2, 5);

   // 매개변수는 함수 몸체 내부에서만 참조할 수 있다.
   console.log(x, y); // ReferenceError : x is not defined
   ```

4. 모든 식별자(변수 이름, 함수 이름, 클래스 이름 등)는,

자신이 **선언된 위치**에 의해 **다른 코드가 식별자 자신을 참조할 수 있는 유효 범위가 결정**된다.

이를 **스코프**라고 한다. 즉, 스코프는 **식별자가 유효한 범위**를 말한다.

5. [예제 13-03]

```
var x = 'global';

function foo() {
    var x = 'local';
    console.log(x); // 1
}

foo();

console.log(x); // 2
```

- **식별자 결정**(identifier resolution)

  : 자바스크립트 엔진은 이름이 같은 2개의 변수 중, 어떤 변수를 참조해야 할 것인지를 결정

- 따라서 스코프란, JS 엔진이 **식별자를 검색할 때 사용하는 규칙**이라고도 할 수 있음.

13.2 스코프의 종류

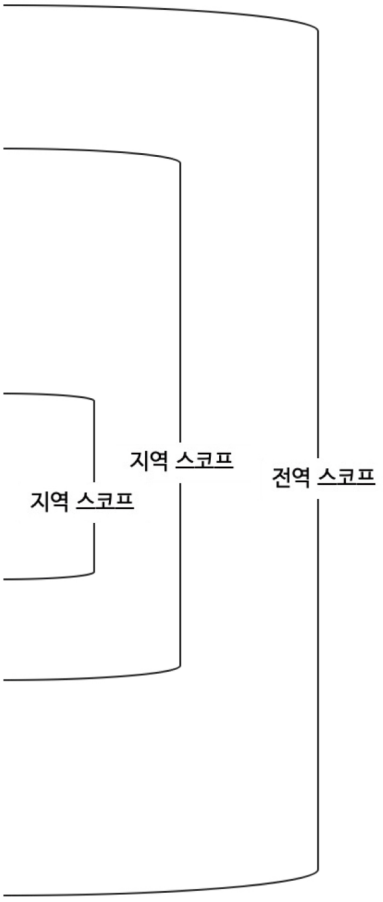
1. 코드는 전역(global), 지역(local)로 구분

구분	설명	스코프	변수
전역	코드의 가장 바깥 영역	전역 스코프	전역 변수
지역	함수 몸체 내부	지역 스코프	지역 변수

13.2.1 전역과 전역 스코프

1. **전역 변수는 어디서든지 참조할 수 있다.**

```
var x = "global x";
var y = "global y";

function outer() {
  var z = "outer's local z";

  console.log(x); // ① global x
  console.log(y); // ② global y
  console.log(z); // ③ outer's local z

  function inner() {
    var x = "inner's local x";

    console.log(x); // ④ inner's local x
    console.log(y); // ⑤ global y
    console.log(z); // ⑥ outer's local z
  }

  inner();
}

outer();

console.log(x); // ⑦ global x
console.log(z); // ⑧ ReferenceError: z is not defined
```



13.2.2 지역과 지역 스코프

1. 지역 : **함수 몸체 내부**를 말한다.

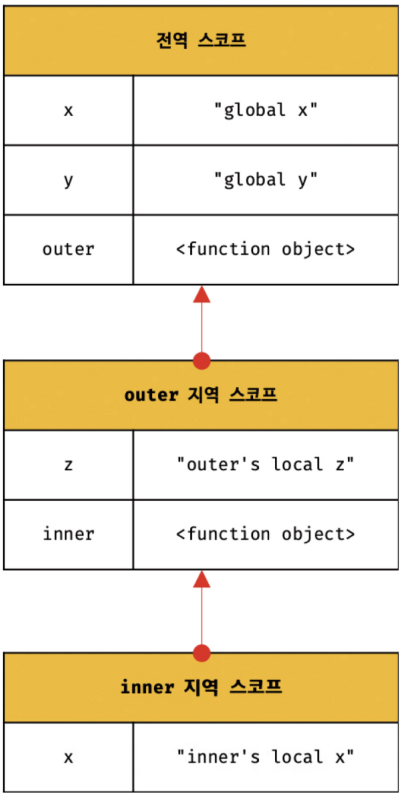
- **지역 변수는 자신의 지역 스코프와, 하위 지역 스코프에서 유효하다.**

13.3 스코프 체인

1. 함수는 중첩될 수 있으므로 함수의 지역 스코프도 중첩될 수 있다.

   → **스코프가 함수의 중첩에 의해 계층적 구조를 갖는다**는 것을 의미

2. **스코프 체인**(scope chain) : 스코프가 계층적으로 연결된 것



3. **변수를 참조할 때 자바스크립트 엔진은,**

   스코프 체인을 통해 변수를 참조하는 코드의 스코프에서 시작하여

   상위 스코프 방향으로 이동하며 선언된 변수를 검색(identifier resolution) 한다.

4. 스코프 체인은 물리적인 실체로 존재

➡️ 자바스크립트 엔진은 위 그림과 유사한 자료구조인 렉시컬 환경을 실제로 생성

13.3.1 스코프 체인에 의한 변수 검색

1. **상위 스코프에서 유효한 변수는 하위 스코프에서 자유롭게 참조할 수 있지만,**
   하위 스코프에서 유효한 변수를 사위 스코프에서 참조할 수 없다.
   ✅. 스코프 체인의 계층적 구조는 **상속**(inheritance)과 유사

13.3.2 스코프 체인에 의한 함수 검색

1. 함수는 식별자 + 함수 객체가 할당된 것 외에는, 일반 변수와 다를 바 X
   스코프 : "변수를 검색할 때 사용하는 규칙" ➡️ **"식별자를 검색하는 규칙"**

13.4 함수 레벨 스코프

1. 지역 - 함수 몸체 내부 & 지역 스코프를 만듦.
   → **코드 블록이 아닌 함수에 의해서만 지역 스코프가 생성된다**는 의미

2. C, JAVA 비롯한 대부분 프로그래밍 언어는 함수 몸체뿐 아니라,
   모든 코드 블록(if, for, while 등)이 지역 스코프를 만든다.
   이러한 특성을 **블록 레벨 스코프**(block level scope)라 한다.

3. **var 키워드로 선언된 변수는 오로지 함수의 코드 블록(함수 몸체)만을 지역 스코프로 인정.**
   이러한 특성을 **함수 레벨 스코프**(function level scope)라 한다.

13.5 렉시컬 스코프

1. [예제 13-09]

```
var x = 1;

function foo() {
    var x = 10;
    bar();
}

function bar() {     // 전역에서 bar()함수 정의
    console.log(x);
}

foo();  // ?
bar();  // ?
```

두 가지 패턴을 예측할 수 있다.

> 1. **함수를 어디서 호출**했는지에 따라 함수의 상위 스코프를 결정.
> → 동적 스코프

> 2. **함수를 어디서 정의**했는지에 따라 함수의 상위 스코프를 결정.
> → **렉시컬(정적) 스코프**